C1_W5_Lab_1_exploring-callbacks

May 10, 2021

1 Ungraded Lab: Introduction to Keras callbacks

In Keras, `Callback` is a Python class meant to be subclassed to provide specific functionality, with a set of methods called at various stages of training (including batch/epoch start and ends), testing, and predicting. Callbacks are useful to get a view on internal states and statistics of the model during training. The methods of the callbacks can be called at different stages of training/evaluating/inference. Keras has available callbacks and we'll show how you can use it in the following sections. Please click the **Open in Colab** badge above to complete this exercise in Colab. This will allow you to take advantage of the free GPU runtime (for faster training) and compatibility with all the packages needed in this notebook.

1.1 Model methods that take callbacks

Users can supply a list of callbacks to the following `tf.keras.Model` methods: * `fit()`, `fit_generator()` Trains the model for a fixed number of epochs (iterations over a dataset, or data yielded batch-by-batch by a Python generator). * `evaluate()`, `evaluate_generator()` Evaluates the model for given data or data generator. Outputs the loss and metric values from the evaluation. * `predict()`, `predict_generator()` Generates output predictions for the input data or data generator.

1.2 Imports

[3]:
```
from __future__ import absolute_import, division, print_function,␣
↪unicode_literals

try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import io
from PIL import Image
```

```
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping,␣
↪LearningRateScheduler, ModelCheckpoint, CSVLogger, ReduceLROnPlateau
%load_ext tensorboard

import os
import matplotlib.pylab as plt
import numpy as np
import math
import datetime
import pandas as pd

print("Version: ", tf.__version__)
tf.get_logger().setLevel('INFO')
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Version:  2.1.0
```

2 Examples of Keras callback applications

The following section will guide you through creating simple Callback applications.

[4]:
```
# Download and prepare the horses or humans dataset

splits, info = tfds.load('horses_or_humans', as_supervised=True,␣
↪with_info=True, split=['train[:80%]', 'train[80%:]', 'test'])

(train_examples, validation_examples, test_examples) = splits

num_examples = info.splits['train'].num_examples
num_classes = info.features['label'].num_classes
```

```
      ␣
↪---------------------------------------------------------------------------

      AbortedError                                 Traceback (most recent call␣
↪last)

      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/utils/
↪py_utils.py in try_reraise(*args, **kwargs)
      467   try:
   --> 468     yield
      469   except Exception as e:  # pylint: disable=broad-except
```

```
      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/load.py␣
↪in builder(name, data_dir, **builder_init_kwargs)
      206     with py_utils.try_reraise(prefix=f"Failed to construct dataset␣
↪{name}: "):
  --> 207       return cls(data_dir=data_dir, **builder_kwargs,␣
↪**builder_init_kwargs)  # pytype: disable=not-instantiable
      208

      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/
↪dataset_builder.py in __init__(self, data_dir, config, version)
      177     else:  # Use the code version (do not restore data)
  --> 178       self.info.initialize_from_bucket()
      179

      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/
↪dataset_info.py in initialize_from_bucket(self)
      437     tmp_dir = tempfile.mkdtemp("tfds")
  --> 438     data_files = gcs_utils.gcs_dataset_info_files(self.full_name)
      439     if not data_files:

      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/utils/
↪gcs_utils.py in gcs_dataset_info_files(dataset_dir)
       83   """Return paths to GCS files in the given dataset directory."""
  ---> 84   return gcs_listdir(posixpath.join(GCS_DATASET_INFO_DIR,␣
↪dataset_dir))
       85

      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/utils/
↪gcs_utils.py in gcs_listdir(dir_name)
       76   root_dir = gcs_path(dir_name)
  ---> 77   if _is_gcs_disabled or not exists(root_dir):
       78     return None

      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/utils/
↪gcs_utils.py in exists(path)
       39   try:
  ---> 40     return tf.io.gfile.exists(path)
       41   # * UnimplementedError: On windows, gs:// isn't supported.
```

```
      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/lib/io/
↪file_io.py in file_exists_v2(path)
      279   try:
  --> 280     pywrap_tensorflow.FileExists(compat.as_bytes(path))
      281   except errors.NotFoundError:


      AbortedError: All 10 retry attempts failed. The last failure:␣
↪Unavailable: Error executing an HTTP request: libcurl code 56 meaning 'Failure␣
↪when receiving data from the peer', error details: Received HTTP code 403 from␣
↪proxy after CONNECT
           when reading metadata of gs://tfds-data/dataset_info/
↪horses_or_humans/3.0.0


  The above exception was the direct cause of the following exception:


      RuntimeError                              Traceback (most recent call␣
↪last)

      <ipython-input-4-7b7d5db780a8> in <module>
        1 # Download and prepare the horses or humans dataset
        2
  ----> 3 splits, info = tfds.load('horses_or_humans', as_supervised=True,␣
↪with_info=True, split=['train[:80%]', 'train[80%:]', 'test'])
        4
        5 (train_examples, validation_examples, test_examples) = splits


      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/load.py␣
↪in load(name, split, data_dir, batch_size, shuffle_files, download,␣
↪as_supervised, decoders, read_config, with_info, builder_kwargs,␣
↪download_and_prepare_kwargs, as_dataset_kwargs, try_gcs)
      339     data_dir = gcs_utils.gcs_path("datasets")
      340
  --> 341   dbuilder = builder(name, data_dir=data_dir, **builder_kwargs)
      342   if download:
      343     download_and_prepare_kwargs = download_and_prepare_kwargs or {}


      /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/load.py␣
↪in builder(name, data_dir, **builder_init_kwargs)
      205   if cls:
      206     with py_utils.try_reraise(prefix=f"Failed to construct dataset␣
↪{name}: "):
```

```
   --> 207       return cls(data_dir=data_dir, **builder_kwargs,␣
↪**builder_init_kwargs)  # pytype: disable=not-instantiable
       208
       209   # If neither the code nor the files are found, raise␣
↪DatasetNotFoundError


       /opt/conda/lib/python3.7/contextlib.py in __exit__(self, type, value,␣
↪traceback)
       128                 value = type()
       129             try:
   --> 130                 self.gen.throw(type, value, traceback)
       131             except StopIteration as exc:
       132                 # Suppress StopIteration *unless* it's the same␣
↪exception that


       /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/utils/
↪py_utils.py in try_reraise(*args, **kwargs)
       468     yield
       469   except Exception as e:  # pylint: disable=broad-except
   --> 470     reraise(e, *args, **kwargs)
       471
       472


       /opt/conda/lib/python3.7/site-packages/tensorflow_datasets/core/utils/
↪py_utils.py in reraise(e, prefix, suffix)
       434     else:
       435       exception = RuntimeError(f'{type(e).__name__}: {msg}')
   --> 436     raise exception from e
       437   # Otherwise, modify the exception in-place
       438   elif len(e.args) <= 1:


       RuntimeError: AbortedError: Failed to construct dataset horses_or_humans:
↪ All 10 retry attempts failed. The last failure: Unavailable: Error executing␣
↪an HTTP request: libcurl code 56 meaning 'Failure when receiving data from the␣
↪peer', error details: Received HTTP code 403 from proxy after CONNECT
            when reading metadata of gs://tfds-data/dataset_info/
↪horses_or_humans/3.0.0
```

[ ]:
```
SIZE = 150 #@param {type:"slider", min:64, max:300, step:1}
IMAGE_SIZE = (SIZE, SIZE)
```

[ ]:
```
def format_image(image, label):
  image = tf.image.resize(image, IMAGE_SIZE) / 255.0
  return  image, label
```

[ ]:
```
BATCH_SIZE = 32 #@param {type:"integer"}
```

[ ]:
```
train_batches = train_examples.shuffle(num_examples // 4).map(format_image).
 ↪batch(BATCH_SIZE).prefetch(1)
validation_batches = validation_examples.map(format_image).batch(BATCH_SIZE).
 ↪prefetch(1)
test_batches = test_examples.map(format_image).batch(1)
```

[ ]:
```
for image_batch, label_batch in train_batches.take(1):
  pass

image_batch.shape
```

[ ]:
```
def build_model(dense_units, input_shape=IMAGE_SIZE + (3,)):
  model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(16, (3, 3), activation='relu',␣
 ↪input_shape=input_shape),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(dense_units, activation='relu'),
      tf.keras.layers.Dense(2, activation='softmax')
  ])
  return model
```

2.1 TensorBoard

Enable visualizations for TensorBoard.

[ ]:
```
!rm -rf logs
```

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir)
```

```
model.fit(train_batches,
          epochs=10,
          validation_data=validation_batches,
          callbacks=[tensorboard_callback])
```

[ ]:
```
%tensorboard --logdir logs
```

2.2 Model Checkpoint

Callback to save the Keras model or model weights at some frequency.

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=5,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[ModelCheckpoint('weights.{epoch:02d}-{val_loss:.2f}.h5',␣
 ↪verbose=1),
          ])
```

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=1,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[ModelCheckpoint('saved_model', verbose=1)
          ])
```

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
```

```
          epochs=2,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[ModelCheckpoint('model.h5', verbose=1)
          ])
```

2.3 Early stopping

Stop training when a monitored metric has stopped improving.

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=50,
          validation_data=validation_batches,
          verbose=2,
          callbacks=[EarlyStopping(
              patience=3,
              min_delta=0.05,
              baseline=0.8,
              mode='min',
              monitor='val_loss',
              restore_best_weights=True,
              verbose=1)
          ])
```

2.4 CSV Logger

Callback that streams epoch results to a CSV file.

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

csv_file = 'training.csv'

model.fit(train_batches,
          epochs=5,
          validation_data=validation_batches,
```

```
          callbacks=[CSVLogger(csv_file)
          ])
```

[ ]:
```
pd.read_csv(csv_file).head()
```

2.5 Learning Rate Scheduler

Updates the learning rate during training.

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

def step_decay(epoch):
        initial_lr = 0.01
        drop = 0.5
        epochs_drop = 1
        lr = initial_lr * math.pow(drop, math.floor((1+epoch)/epochs_drop))
        return lr

model.fit(train_batches,
          epochs=5,
          validation_data=validation_batches,
          callbacks=[LearningRateScheduler(step_decay, verbose=1),
                     TensorBoard(log_dir='./log_dir')])
```

[ ]:
```
%tensorboard --logdir log_dir
```

2.6 ReduceLROnPlateau

Reduce learning rate when a metric has stopped improving.

[ ]:
```
model = build_model(dense_units=256)
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_batches,
          epochs=50,
          validation_data=validation_batches,
          callbacks=[ReduceLROnPlateau(monitor='val_loss',
                                       factor=0.2, verbose=1,
```

```
                                        patience=1, min_lr=0.001),
                    TensorBoard(log_dir='./log_dir')])
```

[ ]:
```
%tensorboard --logdir log_dir
```